STEPS TOWARDS AN OPERATIONAL SENSORS NETWORK PLANNING FOR SPACE SURVEILLANCE

Juan P. Arregui[*] and Juan A. Tejo[†]
GMV, Tres Cantos, 28760, Spain

Carlos Linares López[‡] and Daniel Borrajo[§]
Universidad Carlos III de Madrid, Leganés, 28911, Spain

GMV has been gaining momentum in the application of Mission Planning and Scheduling techniques within the domain of Space Surveillance and Tracking due to the company's involvement in the European Space Situational Awareness programme with the European Space Agency. A *Telescopes Scheduling Simulator* tool has been developed within the scope of the Telescopes Analysis and Design activity which has recently ended with the successful delivery of a configurable telescope network simulator. The tool is capable of performing optimal optical sensor scheduling, from the computation of potential visibilities and observation opportunities, by ingesting and further refining an initial catalogue of man-made objects. It performs its core scheduling process on a rules based inference engine, allowing the user to edit the rules files to optimize planning results. It also provides a performance analysis module to aid the user in the design of an efficient network of global telescopes for SSA. Based on this previous experience, currently on-going activity in collaboration with the Planning and Learning Group (PLG) of Universidad Carlos III de Madrid, *Sensor Planning Services* for the SSA *Prototype Tasking and Data Centres*, is a more ambitious initiative to develop a pre-operational planning system, a precursor service for the future SSA network of sensors which performs the scheduling for ground and space based, optical and radar sensors. Providing a mixed-initiative planning system, it uses a domain-independent automatic planning engine that can be easily extended to accommodate different types of sensors, as scheduling requirements evolve in the future operational phase of the SSA Programme. The purpose of this paper is to expose the technologies behind these two approaches, taking into account the benefits and pitfalls of both and focusing on system performance and scalability to provide a set of conclusions that may be used as design drivers for similar systems.

I. The SSA Sensor Observations Planning problem

THE The purpose of the network of sensors for SSA is to continually observe the sky for man-made and NEO objects to initially build-up and to further maintain the object catalogues with precise orbital information. Two main strategies are used for this purpose: *Survey strategies* and *Tasking strategies*.

A survey is targeted to sweep a specified area of the sky to detect new objects and is the main approach used for initial catalogue build-up and discovery of new objects after a fragmentation event. However, surveys are based on user defined configurations per telescope and do not require a scheduling engine to plan them.

Tasking strategies are used primarily for follow-up catalogue maintenance. This approach focuses on refining and maintaining the orbital information for catalogued objects by revisiting their known orbits. Follow-up is needed throughout the object's lifetime because nominal orbits are altered by various phenomena and the objects disappear from the sky due to orbital decay, conjunction events, etc.

The concept of tasking strategies consists in the definition of activities that drive the observation scheme to be conducted by a prescribed network of sensors. Assuming that the network has been selected to optimize

[*] Project Manager, Mission Planning Systems. jarregui@gmv.com
[†] Head, Mission Planning Systems. jatejo@gmv.com
[‡] Professor, Departamento de Informática, clinares@inf.uc3m.es
[§] University Professor, Departamento de Informática, dborrajo@ia.uc3m.es

observational aspects, the correct scheduling should establish adequate levels of observation loads for each sensor according to the specific capabilities of all involved hardware elements and the expected location of the sensor and the object at any given time. Location is expressed by the following data sets:

- Object ephemerides. This is the predicted state vectors and apparent magnitudes of the object relative to the sensor (e.g. azimuth and elevation). This allows the scheduling of follow-up session on selected catalogued objects.
- Sensor location. For ground-based sensors, this is the geodetic coordinates of the site where the sensor is located. For space-based sensors, this is the state vectors of carrier spacecraft where the sensor is hosted.

Both the design of the tasking strategies and the implementation of the scheduler engine take also into account the following constraints:

- *Weather conditions* at the sensor sites, for ground based sensors.
- *Unexpected failures* (MTBF, MTTR) and times when the system is planned to be under maintenance.
- *Observational constraints* imposed by the minimum horizon elevation, the minimum Sun zenith distance and the presence of the Sun, the Moon (considering its phases) and the galactic plane in the FoV/FoR of sensors.
- *Hardware constraints* regarding the capability to observe certain objects (faintness with respect to sensor sensitivity), rotation and rotation rate limitations, which together with the above mentioned data will drive the priority of each candidate observation by establishing a normalized set of planning decision drivers:
- *Observability*: The capability of an object to be observed by a given sensor. It is constrained by night-time periods, cut-off elevation with respect to the horizon, object illumination (for optical sensors) and minimum spherical angular distance of the FoV/FoR line-of-sight to the moon and galactic plane.
- *Detectability*: The capability of an object to be detected by a given sensor. It is constrained by the sensor's limiting detection magnitude, the object's apparent magnitude (for optical sensors), the Radar Cross Section (for radar sensors), and the sensor's statistical success rate.
- *Accuracy*: The less noise an observation presents, the more accurate it will be. This factor depends on the relative apparent magnitude of the object with respect to the limiting detection magnitude of the sensor. Noisier measurements are obtained from fainter objects or objects with a smaller cross-section. Noisier measurements are also obtained near detection limits.

The production of an optimal plan for a given sensor will comprise a sequence of commands for observations per sensor, taking into account the above mentioned data and constraints. The throughput of the scheduling engine and that of the overall planning system will strongly depend on the planning technology utilized and its ability to express these data and constraints into elements that can be mapped to the scheduling engine's domain. Two different planning techniques have been studied in the context of the SSA "*Telescopes Scheduling Simulator*" and "*Sensor Planning Services*" activities. These techniques are described in the following paragraphs.

From a very general point of view, problem solving techniques can be decomposed in three different groups:

- Inference methods, such as rule-based systems;
- Domain-dependent search algorithms such as heuristic search, or stochastic local search; and finally,
- Domain-independent procedures such as automated planning.

There is a myriad of available techniques but the previous taxonomy will be considered for the sake of argument. The first class techniques, *inference methods*, have the advantage that they usually employ a domain-independent language to describe the domain knowledge, based on rules and, usually, objects. On the negative side, they do not usually implement backtracking or any other mechanism that could be used to revise previous decisions or explore alternative scenarios. This is, these algorithms are said to implement irrevocable policies. However, they can be very useful if the domain at hand shows a high density of solutions and the main driver is to find solutions quickly instead of finding the best solution available from a set of candidates.

Next, *domain-dependent search algorithms* are known to be very efficient for a wide variety of purposes: from optimally solving problems to just finding satisfying solutions in very challenging domains. Most of its high performance comes from the fact that they use refinements particular of the domain at hand that prevents their applicability in other classes of problems even if they are closely related. In this regard, progress in one domain does not necessarily imply a significant improvement in other domains even if the same techniques are applied in more or less the same fashion.

However, it was observed that the underlying algorithms used in a domain-dependent application could be easily generalized. This led, among other ideas, to the construction of Constraint Programming algorithms that can be specialized for a particular domain with particular algorithms. Still, these implementations are highly domain-dependent and a lot of expertise is usually required for coming out with practical solutions ---such as knowing the representation language which is usually coded, how to invoke specific call-backs or how to overload the generation of descendants and so on. The last step in generalization is taken by the idea of *Automated Planning* where the same solver (which is seen as a black box) is expected to solve any problem, even the hardest ones. While the resulting approach is necessarily slower, it is of more practical use and there are a growing number of applications ranging from logistics operations to firefighting, house building, planning Mars rovers, planning satellite maintenance operations, and mission planning just to mention a few. Besides, the generality posed by Automated Planning can be restricted if necessary when creating a class of solvers to behave optimally or near-optimally for a given domain or number of domains which share a number of features, as is the case in SSA. They combine the advantages of inference techniques (domain-independent high-level modeling language) and domain-dependent search techniques (different heuristics and search procedures that allow efficient problem solving). In short, while Constraint Programming provide a convenient path to domain-independent this happens at the programmer level, whereas Automated Planning uses code where domains and problems are redefined at the user level.

Interestingly, *domain-independent techniques* cover the study of all issues related with automated problem-solving: from representational issues, to the selection of the right choices for solving a particular problem and even how to output the solution and/or offering automated means for measuring the quality of alternative solutions ---because the solver did not consider them before or because the end-user is interested in making what-if analysis.

The techniques considered in Automated Planning have a very high expressivity and allow for the definition of very elaborated techniques. In fact, it has been observed that contrary to the observation made by the domain-dependent research community, progress in classical planning has an immediate effect in other types of domain-independent planning such as numerical planning and temporal planning (which are more relevant to the scope of sensor planning) and even more complex paradigms such as *conformant* and/or *contingent* planning.

II. Planning approach for the Telescopes Scheduling Simulator

The Scheduler module for the *Telescopes Scheduling Simulator* has been implemented as an *Inference Engine* based on rules. This solution has the major advantage in its flexibility to model the processing policy based on business rules, which can be modified, expanded or even replaced by other rules without any software modification.

Business rules implement the selected policy (scheduling algorithms) to select a certain telescope for a specific observation based on a set of evaluated conditions.

A rules inference engine is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about the system behavior. These rules, termed productions, are a basic representation found useful in automated planning, expert systems and action selection. Rules inference engines apply to systems that take decisions, which depend on events or a state of some objects. Rules are composed of a set of conditions (if/when) and a set of actions (then).

- The *conditions* part (`when`); conditions drive and implement the mission policy (scheduling strategy). In the condition part several conditions (related with logical operators) can be written. They can be categorized in the following types:
  - *Scheduling conditions*: they model all considerations related to the implementation of the selected allocation strategy during the planning process (e.g. the management of priorities, management of reward functions, etc).
  - *Constraints conditions*: they model all restrictions or constraints to be considered during the allocation process (e.g. physical telescope constraints, exclusion constraints between two different observation types, elapsed time between consecutive observations for the same telescope, etc.).
- The *actions* part (`then`); when the conditions are met, a task object must be defined to contain the telescope allocation and all the information associated to perform the observation. As well, in the execution part several actions can be written.
- Rule *priority*; this is a mechanism that allows the prioritization of the rules production when more than one rule can be triggered at the same time.

Rules inference engines work over two main dedicated memory spaces; the *Working Memory* and the *Agenda*. Working Memory is a space where the elements that characterize the problem are inserted. These elements are facts that can be verified at the instantiation of a given problem. In the case of the *Telescope Scheduling Simulator* the types of elements fall into the domain of the man-made space objects and telescopes.

Agenda is the internal memory space where the rules are loaded. Once the Working Memory and Agenda are populated, the rules are triggered sequentially in this space by applying the condition part of the rules to the existing elements of the Working Memory. If the conditions of a certain rule are met by the elements that exist in the Working Memory, the execution part of the rule is applied and the actions are executed. These actions can be, for instance, the creation of the items (telescope observations) that will build the plan.

The following figure depicts a rule that has been used for a typical testing scenario during the validation of the *Telescopes Scheduling Simulator*.

```
when
    obj: java_CObject( objID:objectId,  grpID:popGroupId,
                       objectCatalogued == true, limitTime:followUpUpdateLimitTime,
                       catTime:catalogueTime, objPriority:objectPriority )
    opp: java_Opportunity(objectId == objID, telID:telescopeId, siteID:siteId)
    not (java_Visibility(objectId == objID))
    telStatus: java_TelescopeStatus(telescopeId == telID && constraintType == "NONE")
then
  # Calculate the priority
    Hours h = Hours.hoursBetween(catTime, limitTime);
    int deltaTime = h.getHours();
    int newPriority = 100 / deltaTime;
    //Create the observation task
    java_Task t = new java_Task(opp);
    t.setTaskType(t.FOLLOWUP);
    t.setTaskPriority(newPriority);
    insert (t)
```

Figure 1: Example of rule used in for the validation of the Teslescopes Scheduling Simulator

This rule is used for assigning a priority (weight) to each task (observation opportunity) depending on the following attributes of the target object:

- *Catalogue time*: Date and Time when the object was last catalogued – and its ephemerides updated.
- *Follow-up update limit time*: Date and Time limit for the object to be re-visited before its ephemerides have become inaccurate and the object is considered to be lost from the catalogue.

According to the *actions* in this rule, the task will receive a higher priority value as the object approaches its limit time for follow-up. It is intended to assure that objects that are close to becoming lost will receive a higher priority – hence, probability - to be allocated for observation by the network of telescopes.

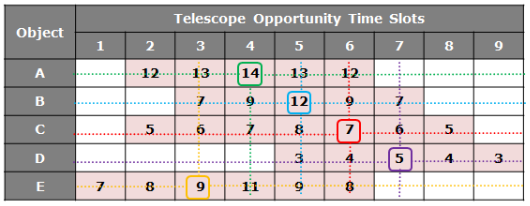
It is worth to mention that the inference engine in the *Telescopes Scheduling Simulator* is used mainly for assigning a priority value to each task based on some prioritization criteria or a combination thereof. In the example above, the update limit time is driving the prioritization. However this system is flexible to adopt any prioritization function represented by rules. Examples of such possible criteria are:

- Whether the telescope is an ESA owned asset or not
- The probability of clouds at the telescope site
- The instruments failure profile

Once the priority for each task is computed, the planning algorithm can generate a final plan based on this prioritization scheme which is now implicit as an absolute figure in the *taskPriority* variable.

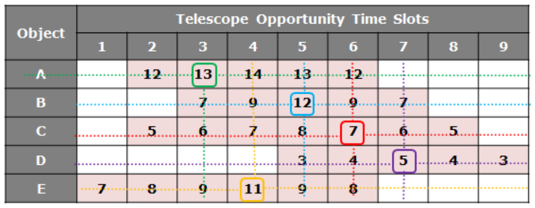
The final step towards production of a plan is the allocation of tasks to free telescope slots. The process starts by finding the highest figure of merit of all object/telescope/slot triples in a three-dimensional matrix. Figure 2is used to describe this procedure for the Telescopes Scheduling Simulator in a simplified procedure for one single telescope. In this example the highest value is “14” and corresponds to the observation of object “A” on the telescope time slot “4”.

Once this pair is fixed, object A and time slot 4 are removed from the matching process. Hence, the next highest figure of merit is now 12 (and not 13, since this value belongs to object A which has already been allocated). Object B is fixed to timeslot 5. The process continues until all objects have been allocated to a telescope time slots or until there aren’t any vacant time slots left.



Object	Telescope Opportunity Time Slots								
	1	2	3	4	5	6	7	8	9
A		12	13	14	13	12			
B			7	9	12	9	7		
C		5	6	7	8	7	6	5	
D					3	4	5	4	3
E	7	8	9	11	9	8			

Figure 2: Allocation of object observations to telescope time slots



Object	Telescope Opportunity Time Slots								
	1	2	3	4	5	6	7	8	9
A		12	13	14	13	12			
B			7	9	12	9	7		
C		5	6	7	8	7	6	5	
D					3	4	5	4	3
E	7	8	9	11	9	8			

Figure 3: Alternative Allocation of object observations to telescope time slots

One of the main benefits of this algorithm is its implementation, which is simple and straight forward. A list of triples (object / telescope / slot), each of which has an absolute priority value computed by the rules engine, is sorted using a standard quick sort algorithm which runs very quickly. The sort process takes a few minutes only because the data first needs to be loaded in the system RAM.

Once the tasks are sorted, the list is processed sequentially until all time slots are used, producing a conflict-free plan which is ready to be ingested by the plan implementation tool to finish the simulation.

The whole process of assigning priority values by the inference engine, sorting the resulting tasks and producing a plan takes only a few minutes to complete, in the worst case, for a scenario of ~ 20K objects and 16 telescopes over a 7 day planning span.

The scheduling algorithm explained in this section is easy to implement and performs well without consuming a large amount of computational resources. Its simplicity, while making it suitable for a telescope network analysis tool and simulator, however lacks the precision needed for the optimization of resource usage in a real tasking scenario, where the amount of sensors available for observations is scarce and expensive to use.

The main drawback of this algorithm is that it does not pursue to globally optimize the results of the plan generation. Using a best-first search approach and an irrevocable policy, the tasks with higher priority are allocated first, but this does not guarantee that the global figure of accumulated priorities is optimized. To graphically understand this issue, Figure 3 shows an alternative plan where the first allocation is object A to time slot 3 (with a priority value of 13 which is not the maximum). This will allow the third allocation to be object E to time slot 4, with a priority of 11. At the end, this approach allows a global accumulated figure of merit of 48, whereas the described implementation above reaches only 47.

The planning approach for the *Sensor Planning Service* activity aims to improve the approach presented for the *Telescopes Scheduling Simulator*.

III. Planning approach for the Sensor Planning Service

Automated Planning is one of the most prolific areas in Artificial Intelligence (AI) whose applicability to real domains has been significantly boosted over the last decades. In particular, there are a growing number of applications built over the idea of either domain-dependent or domain-independent solvers. Note that the word domain is used along this section as the particular scenario where the planning problem is solved. For *Sensor Planning Service* the domain is the planning of the sensors tasks.

Automated Planning is concerned with all issues that arise when creating automatic solvers that are domain-independent. Interestingly, the techniques developed so far can be further specialized to solve problems in particular domains more efficiently. A full description of the capabilities of Automated Planning goes beyond the scope of this paper. Therefore, we content ourselves with discussing some representational issues and to introduce the main trend in Automated Planning research which consists of heuristic search.

A. Representational Problems

Since its conception in 1998, PDDL (Planning Domain Description Language) has become the standard in the scientific community for the representation and exchange of planning domain models and problems. PDDL is a declarative description language that embodies all the definitions that usually appear at any Planning problem. Automated planners require two inputs for solving a given task: a *domain* description that defines the available actions; and a *problem* description that defines a particular task to be solved in that domain. The problem description includes an initial state (with the initial assignment to planning variables), a set of goals, and possibly a metric to be optimized. Thus, PDDL provides two parts: a domain standard definition language and a problem standard definition language. Besides, it has been severely extended so that it can also deal with concepts drawn from the Scheduling community. The current version of PDDL is 3.1.[1]

B. Heuristic Planning

Problem solvers *search* for solutions: they try to find a connection sequence (plan) from a start position to an end position, where start, end, and connections are designed in a way so that the connecting sequence corresponds to the solution of the problem. Finding the connecting sequence involves exploring the space of possible connection choices. The simplest example for such a search is state space search, as used in a number of automated planners such as Fast Forward (FF)[2], Fast Downward[3,**] , or, more recently, LAMA[4,††]: there, the start is the initial state of the system (the current situation), the end is the goal, and the connection choices are actions

A *heuristic function* guides the search. Concretely, it is a piece of program code that, given the description of a search state, returns an integer number meant to estimate the *remaining cost* of the search state: each action has a cost (constantly 1 in the simplest case), and the remaining cost is the minimum summed up cost of actions necessary to reach a state satisfying the goal. Search then prefers those states with lower cost estimates. If the estimates are *informative* - if the heuristic computation does capture important aspects of the problem - then preferring states with lower estimates can dramatically shorten the time needed to find a good solution. There are various standard search methods - methods defining how the heuristic information is exploited - some of which guarantee that the first found solution is optimal, provided the heuristic function does not overestimate the effort to reach the goal state.

Heuristic search planning adopts this method. The main challenge is to define the heuristic functions: since the solvers shall be general, the heuristic functions must be created automatically from the description of the problem. The main trick is to *relax* the problem, i.e. to modify the problem definition in such a way that makes it easier to solve it. The relaxed problem is then solved in every search state, and the cost of the relaxed solution (e.g. the number of actions it uses or the cumulative cost of all the actions) provides the heuristic estimate.

There are a wide number of extensions to this paradigm. In particular, the following section considers the application of sequential satisficing and temporal planning for the implementation of the SSA *Sensor Planning Service*.

IV. Comparison of Automated Planning approaches

In this section, we describe the preliminary empirical evaluation that has been performed for this particular problem. The solver has to find a feasible conflict-free assignment of sensors (both in space and on earth) to objects in space such that a number of constraints are met. The following constraints are particularly interesting: a sensor tracks an object only when the object falls in the acquisition opportunity of the sensor and the sensor is available; a sensor can only track one particular object at a time; besides, all objects have to be tracked. Other than these, there are also a number of constraints that are particularly interesting in the context of the SSA *Sensor Planning Service*. Finally, the scheduling is characterized by a number of parameters that define their utility. This value results from the linear combination of a number of parameters that are used to model the priority of the sensors, the priority of the planning request and the priority of the observation opportunity. All of these parameters are used to score every particular plan according to a linear combination of them that can be either chosen from a list through a HMI or hand-edited by the user.

As a result, the following parameters have been identified as being particularly relevant to the overall performance of the planner:

- *Sensors*: the number of sensors available during the whole makespan is one of the main parameters that affect the overall running time. In general, the number of sensors might be lower than the number of objects to track ---i.e., objects have to compete for the sensors instead of the other way around. Thus, they can be seen as providers. We have conducted experiments with both kinds of cases (more sensors than objects and more objects than sensors).
- *Objects*: the number of objects is equally important and it can be seen as a consumer in a typical scheduling problem.
- *Intervals*: this parameter is defined as the number of observation opportunities per sensor and object. As a matter of fact, it is assumed throughout this document that the intervals, when defined, result from the intersection of the observation opportunity and also the sensor availability. Intervals are twice important: first, if the number increases the solver is forced to consider more alternatives when minimizing a particular objective function; second, even if the number is small, they can be arranged in such a way that the problem becomes very difficult (e.g., there might be so many intersections among them that the solver engine would be forced to backtrack many times for trying all the feasible assignments in a particular time slot)

[**] http://www.fast-downward.org/

[††] the original version being accessible from http://www.informatik.uni-freiburg.de/~srichter/

- *Time slots*: an alternative to intervals consists on defining specific time slots where observations can be scheduled. The same discussion for intervals applies here.
- *Makespan*: it is defined as the time horizon to consider. While it is not a critical parameter in temporal planning, it increases the number of instantiations if sequential satisficing planners are used (see the discussion below). However, their study is also relevant when using temporal planners: the larger the makespan, the more time intervals have to be considered by the planner

In this paper we address all of these issues with two different approaches in heuristic Automated Planning: *sequential satisficing* and *temporal planning*. From a general point of view, general deliberative scheduling deals with the assignment of time labels to a set of actions (either their start and end times, intervals at which they can be executed, or specific time slots assigned to each action). Actions might be previously constrained among each other, so that a given criteria is optimized (e.g., maximum quality of observations, the temporal makespan is minimized, …). In order to solve a real problem we need to find a set of actions (a planning problem) that have temporal constraints among them, because of the temporal relationships among actions (both a planning and a scheduling problem) or because of the existing constraints among the resources used by the actions (typically a scheduling problem).

In the first case (sequential satisficing planning), a number of actions are defined in the domain file that are instantiated with the parameters provided in the problem file - mainly the sensors, objects and time slots. The goal of sequential satisficing is to find an order of actions that transforms the initial state into the goal state - both described in the problem file. This is, by far, the most popular approach in Automated Planning and it has experienced a significant progress during recent years as discussed above. However, time is explicitly considered so that the same action with the same parameters (i.e., sensors, objects and intervals) has to be instantiated, one per each particular time slot when it can be executed. A number of experiments are provided in this document using this technology.

In the second case (temporal planning), time is implicitly considered by the solver engine so that actions are instantiated once for a particular combination of its parameters. Also, preliminary experiments with a temporal planner are provided here.

A. Sequential Satisficing Planning

The goal of this analysis is to understand how pure STRIPS planning scales in a simplified object-planning task. We defined a domain model in a PDDL file that can be found in Figure 4. We first define the name of the domain, ssa-dcii, and some requirements for the planners in order to be able to process this definition (STRIPS planning, together with types of instances, and numerical functions, fluents). Then, we define the types of this domain, object, sensor and slot. Next, predicates (Boolean values) and functions (usually numerical values, but they can also be nominal values) are defined. Finally, we define all actions; in this case only one action is needed that assigns a sensor to observe an object at a given time slot. Given that most planners deal with minimizing criteria, and we would like to maximize quality of observations (that takes into account the probability of the sensor to be working at that time slot, the weather conditions, the appropriateness of using it for observing the object and any other relevant parameter), we increase a counter (total-quality) with one minus the corresponding quality, where the quality has been defined between 0 and 1. Consider that the goal of this technical note is to analyze complexity and scalability aspects of the domain, rather than providing the exact models that will be defined later on.

```
(define (domain ssa-dcii)
  (:requirements :strips :typing :fluents)
  (:types object sensor slot)
  (:predicates
     (observed ?o - object)
     (available ?s - sensor ?i - slot)
     (visible ?s - sensor ?o - object ?i - slot))
  (:functions
     (quality-object ?s - sensor ?o - object ?i - slot)
     (total-quality))
  (:action observe
          :parameters (?s - sensor ?o - object ?i - slot)
          :precondition (and (available ?s ?i)(visible ?s ?o ?i))
          :effect (and (observed ?o)
            (not (available ?s ?i))
            (increase (total-quality) (- 1 (quality-object ?s ?o ?i))))))
```

Figure 4: Domain model in PDDL (sequential satisficing).

1. *Influence of the number of objects to track*

Then, in the first experiment we defined several problems of increasing complexity by using a random problem generator. Those problems had 110 sensors, an increasing number of objects from 10 to 90 in steps of 10, and 720 time slots (5 seconds per slot and one hour of time span). We generated the problems in such a way that only one sensor can follow an object at only one time slot. In Figure 5 we include a fragment of such problem files in PDDL with 10 objects. We first define the problem and domain names and the specific constants to be used in the problem: 10 sensors, 10 objects and 10 time slots. Since these are the only sensors that can follow any object and those are the time slots where an observation can occur, we only need to define those numbers of sensors and time slots in the problem. But those were taken randomly from a set of 110 sensors and 720 time slots. We then need to define the initial state: all objects that are visible by any sensor at any time slot; all sensors that are available at any time slot, and the expected quality of observing an object with a sensor at a given time slot. Finally we have to specify the goals (all objects are observed by at least one sensor) and possible a metric criteria (minimizing total-quality, which is the same as explained before as maximizing quality).

```
(define (problem prob-110-10-720) (:domain ssa-dcii)
  (:objects s49 s5 s29 s89 s104 s75 s16 s57 s4 s108 - sensor
            o9 o8 o7 o6 o5 o4 o3 o2 o1 o0 - object
            t302 t564 t544 t80 t388 t468 t526 t7 t35 t134 - slot)
  (:init
       (visible s49 o9 t302) (available s49 t302)
       (= (quality-object s49 o9 t302) 0.59)
       (visible s5 o8 t564) (available s5 t564)
       (= (quality-object s5 o8 t564) 0.63)
           . . .
       (visible s108 o0 t134) (available s108 t134)
       (= (quality-object s108 o0 t134) 0.81)
       (= (total-quality) 0))
  (:goal (and (observed o9) (observed o8) (observed o7) (observed o6)
                (observed o5) (observed o4) (observed o3) (observed o2)
             (observed o1) (observed o0)))
  (:metric minimize (total-quality)))
```

Figure 5. An example problem file in PDDL (sequential satisficing).

All generated problems are guaranteed to be solvable (in fact we provide a potential solution in the problem file for validation purposes).

Now, we run two planners and we plot here the time (in seconds) taken to solve the problems. In abscises we show the number of objects considered and in the ordinates, the time in seconds. We see that the problems become exponentially harder to solve when the number of objects increases. We have used the planner LPG[5] and the planner Sayphi[6] with two different search techniques: Enforced Hill Climbing (EHC), and A*. EHC is a greedy technique, so it does not assure us to obtain the optimal solution. And A* only assures us to obtain the optimal solution if the heuristic is admissible (its predicted value is less than or equal the optimal value for each node in the search tree). Given that most heuristics used in satisficing planning are non-admissible, this instantiation of A* does not assure either optimality, but usually will return better values for the metric criteria than the solutions obtained by EHC. In this specific case, where there is only one possible sensor per object and time slot, the solution quality of the solutions returned by both techniques will be the same, since they are the same solutions (they might be permutations of the other, which in the case of this domain, consists of the same solution).

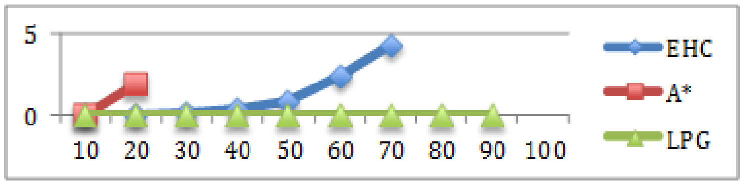


Figure 6. Time (in seconds) to solve problems of increasing number of objects.

As we can see, EHC is only able to solve up to 70 objects. Starting from 80 objects up it runs out of memory (6Gb). A* is only able to solve up to 20 objects before reaching the memory bound. In this test, LPG was the best alternative.

2. *Influence of the number of sensors per object to track*

In the second experiment, we generated a new set of problems where we allow up to two sensors to follow each object. And we performed the same experiments. Next, we show the results.

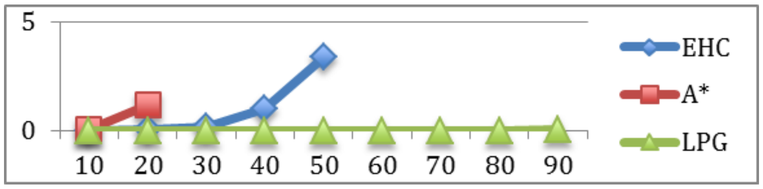


Figure 6. Time (in seconds) to solve problems of increasing number of objects when up to two sensors can follow each object.

As we can see, EHC is only able to solve problems with up to 50 objects and A* 20 objects. Again, LPG is the best option. We can also see that the complexity increases with respect to having only one sensor per object, since the branching factor (number of potentially applicable actions at each node in the search) increases; there are more sensors in the problem definition, more observation opportunities, and more time slots. If we have *S* sensors, *O* objects and T time slots, in theory we could have a branching factor of around *SxOxT*. Given that finding the optimal solution could require in the worst case exploring all nodes, the expected number of nodes to be explored will be:

$$Nodes = \sum_{i=0..d} (SxOxT)^d$$

where *d* is in this case the number of objects to be observed. Given that usually time to search is proportional to the number of nodes explored, we will need in theory a time that grows exponentially.

We repeated the same experiments with 5 sensors per object, and EHC could only solve problems of up to 30 objects and A* again up to 20 objects. This increase in difficulty comes again from the size of the branching factor caused by having more alternative sensors per object.

3. *Influence of the number of time span*

The next experiment consisted on fixing 110 sensors and 200 objects and increasing the number of time slots from 1 day (17280 time slots of 5 seconds each) up to 30 days. We allowed up to 2 sensors per object. The results in time to solve are shown in Figure 5, where we only show the results using LPG.

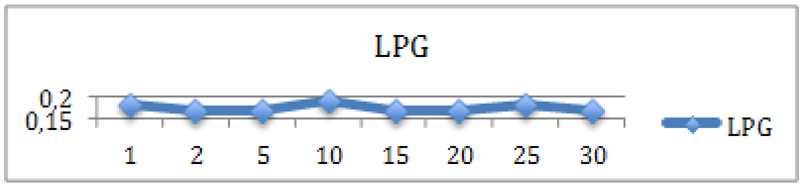


Figure 7. Time (in seconds) to solve problems of increasing number of days when up to two sensors can follow each object.

We then fixed 110 sensors and 30 days and increased the number of objects from 100 up to 800. LPG has an internal variable of maximum number of goals set to 250, so it could not solve problems starting with 300 objects (since there is one goal per object).

B. Temporal Planning

The temporal definitions, as needed here, are compliant with version 2.2 of the standard PDDL[7]. As usually, a planning task is defined both by a domain file and a problem file. The following domain file shows the definition of the only action considered in all the following experiments: *acquire*

```
(define (domain ssa-domain)
 (:requirements :typing :timed-initial-literals :negative-preconditions
                :durative-actions)
 (:types sensor spobject interval)
 (:predicates (visibility ?s - sensor ?o - spobject ?i - interval)
              (available ?s - sensor)
          (acquired ?o - spobject)
          (open ?i - interval))
 (:durative-action acquire
       :parameters (?s - sensor ?o - spobject ?i - interval)
       :duration (= ?duration 2)
       :condition (and (at start (open ?i))
              (at end (open ?i))
              (at start (visibility ?s ?o ?i))
              (at start (available ?s)))
       :effect (and (at start (acquired ?o))
           (at start (not (available ?s)))
           (at end (available ?s)))))
```

Figure 7. Domain model in PDDL (Temporal Planning).

As it can be seen, this domain file defines a particular domain named *ssa-domain*. It consists only of a single action that tracks a particular object (*?o*) with a particular sensor (*?s*) sometime within a particular interval (*?i*). The action consists of four parts:

- *Parameters*: it defines the parameters accepted by the action
- *Duration*: in temporal planning, actions are said to be durative, i.e., the planner is aware that their execution lasts for some time and it guarantees that their execution does not have any conflicts with any other one.
- *Conditions*: define the predicates that have to be true for the action to take place. It is particularly important to note that the conditions are qualified with keywords such as "at start" and "at end". These expressions require the condition to be true at the beginning of the execution of the action or at the end, respectively. Besides, "during" is recognized also by the current standard of PDDL.
- *Effect*: define the predicates that affect the world. If they are preceded by not then the corresponding predicate is deleted. Otherwise, they are added.

```
define (problem ssa-problem-1)
(:domain ssa-domain)
(:objects
 sensor1 sensor2 - sensor
 object1 object2 - spobject
 interval1-1 interval1-2 interval2-1
 interval2-2 - interval)
; start state
(:init
 ; availabilities
 (available sensor1)
 (available sensor2)
 ; visibilities
 (visibility sensor1 object1 interval1-1)
 (visibility sensor1 object2 interval1-2)
 (visibility sensor2 object1 interval2-1)
 (visibility sensor2 object2 interval2-2)
 ; observation opportunities
 (at 53 (open interval1-1))
 (at 55 (not (open interval1-1)))
   . . . . . .
 (at 3 (open interval2-1))
 (at 6 (not (open interval2-1)))
 (at 81 (open interval2-1))
 (at 85 (not (open interval2-1)))
   . . .
 (at 80 (open interval2-2))
 (at 83 (not (open interval2-2)))
)
; goal state
(:goal
 (and
  ; fulfilled observations
  (acquired object1)
  (acquired object2)
 ))
(:metric minimize (total-time)))
```

Figure 8. An example problem file in PDDL (temporal planning)

On the other hand, a planning task is fully defined only when a problem file is provided as well. In particular, problem files define all the constants in the current problem and both the initial state and the goal state. Figure 8 shows a fragment of a very simple example with two objects and two sensors that can track the objects in two different intervals but the second object has up to four different opportunities to follow the second object. As shown at the bottom of the problem file, the objective function is to minimize the total-time or, alternatively, the makespan of the resulting schedule. For the scalability analysis, no experiments have been performed with other metrics.

It is worth noting that in this particular example, the observation opportunities are defined with intervals whose name follows the following convention: *interval-s-o*, where *s* stands for a sensor identifier and *o* is the identifier of a particular object. The different time intervals at which the sensor can track the object are defined with Timed Initial Literals (TIL) that are the resource provided by PDDL 2.2 to define static exogenous events.

Initially, all temporal planners taking part in the temporal track of the Seventh International Planning Competition where considered. There were, in total, 12. However, only three support Time Initial Literals (TILs). These planners are: TLP-GP[8], SHARAABI[9] and POPF2[10]. The first two were not chosen for the empirical evaluation of the scalability analysis because they performed rather poorly with small-sized examples. However, an improved version of POPF2, POPF3 provided a reasonably good performance.

In the following, we examine the importance of all the aforementioned parameters using always the planner POPF3[‡‡] with a number of examples randomly generated. In all cases, the number of sensors is lower than the number of objects and all problems are guaranteed to be solvable. Besides, all the intervals have been generated with a different width that is randomly chosen in the interval [5, 90).

1. *Influence of the number of objects to track*

In this case, 18 different planning tasks were randomly generated. All of them considered 20 sensors and a varying number of objects from 50 to 100 in steps of 10. The makespan was fixed to 86,400 units that correspond to a full day with a granularity of 1 second each. Besides, every sensor is given, in the mean, 50 observation opportunities per object ---which results here in up to 20x100x50=100,000 different intervals. For every combination of the number of sensors, objects and intervals, three planning tasks were generated. Figure 9shows the distribution of the overall running time for the same set of experiments reported in the previous table. Every point represents the mean value of the time (in seconds) for all problems with the same combination of sensors, objects, makespan and intervals.

Pr. Id	Sensors	Objects	Makespan	Intervals	Time (sec.)
0	20	50	86400	50	20,29
1	20	50	86400	50	17,77
2	20	50	86400	50	60,84
3	20	60	86400	50	56,88
4	20	60	86400	50	72,82
5	20	60	86400	50	81,55
6	20	70	86400	50	99,8
7	20	70	86400	50	98,43
8	20	70	86400	50	101,87
9	20	80	86400	50	105,37
10	20	80	86400	50	19,45
11	20	80	86400	50	28,75
12	20	90	86400	50	29,11
13	20	90	86400	50	28,93
14	20	90	86400	50	42,59
15	20	100	86400	50	42,35
16	20	100	86400	50	42
17	20	100	86400	50	57,25

Table 1:Benchmark 1: 18 planning tasks, 20 sensors, 50-100 objects, 50 intervals per object

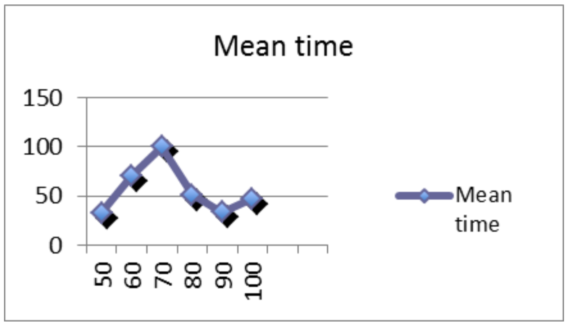


Figure 9: Mean time execution for benchmark 1: Benchmark 1: 18 planning tasks, 20 sensors, 50-100 objects, 50 intervals per object

As it can be seen, the time does not necessarily increase with the number of objects. The reason is that even if the number of objects increase, the problem might be simpler if the intervals (randomly generated in these cases) are arranged in such a way that the number of feasible combinations decreases - see discussion below

2. *Influence of the number sensors*

In the second set of experiments, 15 different planning tasks were randomly generated. In all cases, the number of objects is fixed to 60 and the number of intervals is, as before, 50 per pair (sensor, object) randomly distributed over a makespan of 86400 time units. The number of sensors ranged from 10 to 50 in steps of 10 ---note that we intentionally stopped before the number of sensors equals the number of objects. Figure 10 shows the distribution of the mean time for all the sensors considered in these experiments. Each point is computed as the mean of all problems with the same number of sensors

[‡‡] POPF3 has been originally designed and coded by Andrew Coles, Amanda Coles, Maria Fox and Derek Long from the King's College London (UK). Besides, Andrew Coles has kindly assisted during the first stages of the empirical evaluation documented here.

Pr. Id	Sensors	Objects	Makespan	Intervals	Time (sec.)
0	10	60	86400	50	25,38
1	10	60	86400	50	27,52
2	10	60	86400	50	36,58
3	20	60	86400	50	36,25
4	20	60	86400	50	39,98
5	20	60	86400	50	41,77
6	30	60	86400	50	38,88
7	30	60	86400	50	24,45
8	30	60	86400	50	28,49
9	40	60	86400	50	28,24
10	40	60	86400	50	29,38
11	40	60	86400	50	31,98
12	50	60	86400	50	32,52
13	50	60	86400	50	36,27
14	50	60	86400	50	33,79

Table 2:Benchmark 2: 15 planning tasks, 10-50 sensors, 60 objects, 50 intervals per object

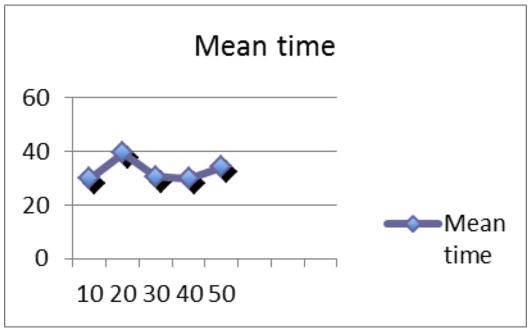


Figure 10: Mean time execution for benchmark 2: Benchmark 1: 15 planning tasks, 10-50 sensors, 60 objects, 50 intervals per object

As it can be seen, the overall runtime seems to be rather robust to the number of sensors, even after multiplying by 5 its availability. As a matter of fact, the problem becomes increasingly simpler as the number of sensor increases - and this is the reason why the maximum mean time happens for a rather low number of sensors, 20

3. Influence of the number of intervals

As mentioned above, the number of intervals is very relevant to the overall complexity of the planning task to solve. In this case, the number of sensors and objects was fixed to 20 and 100 respectively. Twenty one different problems were randomly generated with a number of intervals ranging from 10 to 70 per (sensor, object) in steps of 10 during a makespan equal to 86,400 time units. As in the previous cases, Figure 12 shows the distribution of the overall running time as the number of intervals grows from 10 to 70, where each plot represents the mean of all the problems with the same combination of the input parameters.

Pr Id	Sensors	Objects	Makespan	Int.	Time (sec.)
0	20	100	86400	10	27,35
1	20	100	86400	10	32,27
2	20	100	86400	10	81,18
3	20	100	86400	20	97,1
4	20	100	86400	20	95,08
5	20	100	86400	20	94,26
6	20	100	86400	30	104,76
7	20	100	86400	30	120,65
8	20	100	86400	30	122,11
9	20	100	86400	40	114,37
10	20	100	86400	40	124,05
11	20	100	86400	40	127,51
12	20	100	86400	50	28,33
13	20	100	86400	50	118,01
14	20	100	86400	50	50,99
15	20	100	86400	60	54,07
16	20	100	86400	60	50,56
17	20	100	86400	60	74,32
18	20	100	86400	70	67,4
19	20	100	86400	70	73,79
20	20	100	86400	70	85,54

Table 3:Benchmark 3: 21 planning tasks, 20 sensors, 100 objects, 10-70 intervals per object

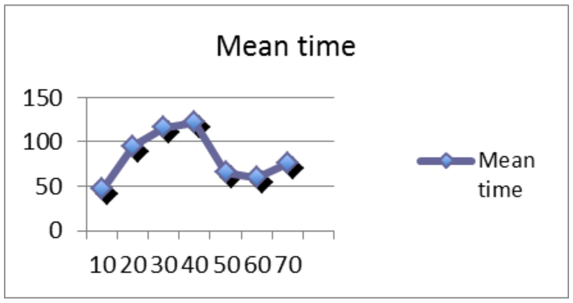


Figure 11: Mean time execution for benchmark 3: Benchmark 1: 21 planning tasks, 20 sensors, 100 objects, 10-70 intervals per object

Again, the running time does not necessarily increase with the number of intervals and the maximum is observed for a mean number of intervals of 40 per sensor and object (which, in this case means that the total number of intervals defined is 20x100x40= 80,000). Again, the reason is that the arrangement of the observation opportunities is even more important than their number. In any case, it can be seen that the time does not grow exponentially and that there is not a very large variance in the results - especially if the very simple cases with 10 intervals are removed from the case study

4. *Influence of the makespan*

Finally, the makespan is also considered in this study. The main goal of the study is to see if extending the makespan affects the overall running time or not. In all cases, the number of sensors and objects were fixed to 20 and 100 respectively and up to 50 random observation opportunities were generated per sensor and object. The makespan ranged from 1 to 7 days with a precision of 1 second - i.e., it ranges from 86400 to 604800 time units in steps of 86400. The larger makespan is equivalent to two weeks with a granularity of 2 seconds. Figure 12 shows the mean overall running time for all those cases shown in the previous table with the same combination of the relevant parameters

Pr. Id	Sensors	Objects	Makespan	Intervals	Time (sec.)
0	20	100	86400	50	113,07
1	20	100	86400	50	105,21
2	20	100	86400	50	156,64
3	20	100	172800	50	149,17
4	20	100	172800	50	150,17
5	20	100	172800	50	149,48
6	20	100	259200	50	153,85
7	20	100	259200	50	163,06
8	20	100	259200	50	153,99
9	20	100	345600	50	160,52
10	20	100	345600	50	172,09
11	20	100	345600	50	157,32
12	20	100	432000	50	112,58
13	20	100	432000	50	157,92
14	20	100	432000	50	135,13
15	20	100	518400	50	122,33
16	20	100	518400	50	132,53
17	20	100	518400	50	125,31
18	20	100	604800	50	134,41
19	20	100	604800	50	136,03
20	20	100	604800	50	143,45

Table 4:Benchmark 4: 21 planning tasks, 20 sensors, 100 objects, 50 intervals per object

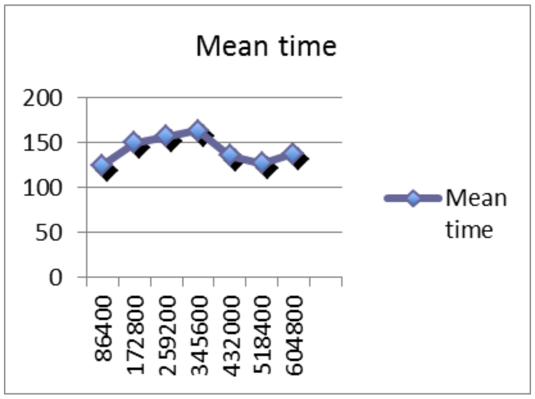


Figure 12: Mean time execution for benchmark 4: Benchmark 1: 21 planning tasks, 20 sensors, 100 objects, 50 intervals per object

As expected, the makespan does not have any significant impact in the running time and the resulting figure shows a linear profile

V. Conclusion

This paper has described the analysis of different technologies used for sensor planning in the scope of the SSA Sensor Scheduling Simulator and Sensor Planning Service activities with ESA. The following topics were addressed:

- Planning of a network of Telescopes using prioritization based on a rules engine
- Planning of a network of Sensors with using prioritization based on Sequential Satisficing Automated Planning
- Planning of a network of Sensors with using prioritization based on Temporal Automated Planning

The results of this study will serve as design drivers for the final implementation of the SSA Sensor Planning Service and can be used as for the design of similar systems by the planning community.

Appendix A
Acronym List

AI	Artificial Intelligence
EHC	Enforced Hill Climbing
FoR	Field of Regard
FoV	Field of View
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
NEO	Near-Earth Object
PDDL	Planning Domain Definition Language
RAM	Random Access Memory
SSA	Space Situational Awareness
TIL	Timed Initial Literals

References

[1]Daniel L. Kovacs., "BNF Definition of PDDL 3.1." 2011. URL: http://www.plg.inf.uc3m.es/ipc2011-deterministic/Resources

[2]J. Hoffmann, B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," *Journal of Artificial Intelligence Research*, Vol. 14, 2001, pp. 253-302

[3]M Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, 2006

[4]Silvia Richter and M. Westphal, "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks," *Journal of Artificial Intelligence Research,* Vol. 39, 2010, pp. 127-177

[5]A. Gerevini, A. Saetti and I. Serina, "Planning through Stochastic Local Search and Temporal Action Graphs," *Journal of Artificial Intelligence Research*, Vol. 20, 2003, pp. 239-290

[6]T. de la Rosa, A. García-Olaya and D. Borrajo, "Using Cases Utility for Heuristic Planning Improvement, Case-Based Reasoning Research and Development," *7th International Conference on Case-Based Reasoning*, 2007, pp. 137-148

[7]Stefan Edelkamp and Jörg Hoffmann. "PDDL 2.2: The Language for the Classical Part of the 4th International Planning Competition." Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2004

[8]F. Maris and P. Régnier, "A Planner to Solve Temporally-Expressive Problems and to Exploit Floating Solution-Plans" *The 2011 International Planning Competition*, eds. Ángel García Olaya, Sergio Jiménez and Carlos Linares López, 2011, pp. 132-136, URL: http://e-archivo.uc3m.es/handle/10016/11710

[9]Bharat Ranjan Kavuluri, "Extending Temporal Planning for the Interval Class." *The 2011 International Planning Competition*, eds. Ángel García Olaya, Sergio Jiménez and Carlos Linares López, 2011, pp. 79-82, URL: http://e-archivo.uc3m.es/handle/10016/11710

[10]Amanda Coles, Andrew Coles, Maria Fox and Derek Long. "POPF2: a Forward-Chaining Partial Order Planner", *The 2011 International Planning Competition*, eds. Ángel García Olaya, Sergio Jiménez and Carlos Linares López, 2011, pp 65-70, URL: http://e-archivo.uc3m.es/handle/10016/11710[9]